Exercise 4

Local Explanations

Local interpretable model-agnostic explanations (LIME) and Anchors are powerful tools to explain local regions. While LIME trains a local (interpretable) surrogate model to imitate the model's behavior, Anchors explain instances by areas with high precision and coverage. This exercise aims to gain more insight into how the methods work and to sensitize your awareness of potential problems.

- You can get a bonus point for this exercise if you pass at least 85% of the tests. Code is automatically tested after pushing. If your tests fail, either your code is wrong, or you solved the task differently. In the second case, we will manually check your solution.
- Three collected bonus points result in a 0.33 increase of the final grade.
- You are allowed to work in groups with up to three people. You do not need to specify your ErgebnisPIN as it is already collected from the entrance test.
- Follow the 'README.md' for further instructions.
- Finish this exercise by 10th November, 2021 at 11:59 pm.

---

1 Local Interpretable Model-Agnostic Explanations

In the following, you are guided to visualize LIME to interpret a Multi-Layer Perceptron. We use two features and explore LIME on a multi-classification problem.

Associated file: *tasks/custom_lime.py*.

1.1 Prepare Mesh Data

First of all, we prepare data to visualize the feature space. Theoretically speaking, we create a $N \times N$ grid, and every point in this grid is associated with a value. This value is obtained by the model's predict method.

Span a linear space for x_i and y_i with $0 \leq i \leq N$ in the method *get_mesh*. The first feature is associated with the x-axis, whereas the second feature is to the y-axis. Ignore all other features. Consider the lower and upper bounds using the configspace and ensure the values z are in the correct shape. Equation 1 shows the shape more precisely. $\mathcal{M}(x_i, y_j)$ is the prediction from the i-th x value and the j-th y value.

$$z = \begin{bmatrix} \mathcal{M}(x_0, y_0) & \dots & \mathcal{M}(x_N, y_0) \\ \vdots & \ddots & \vdots \\ \mathcal{M}(x_0, y_N) & \dots & \mathcal{M}(x_N, y_N) \end{bmatrix} \tag{1}$$

1.2 Plot Color Mesh

After calculating the color values, we want to apply the values in the function *plot_mesh*. Use *pcolormesh* with *viridis* as colors and make sure you deactivate the grid.

1.3 Sample Points

Next, we sample points, which are later used to train the local surrogate model. Complete *sample_points* by randomly sampling from a uniform distribution. Consider the lower and upper bounds from the configspace again. Having a detailed look into the API will save you a lot of time.

1.4 Weight Points

Given a selected point $\hat{x}$ and the sampled points X from the previous task, we now want to weight the points. Use the following equation with d as Euclidean distance to calculate the weight of a single point $x_i \in X$:

$$n(x_i) = exp(-d(\hat{x}, x_i)^2/\sigma^2). \tag{2}$$

To make plotting easier later on, the weights should be normalized between zero and one. Finally, return the normalized weights in *weight_points*.

1.5 Plot Points

Complete *plot_points_in_mesh* by adding the sampled (weighted) points into the plot. Make sure that both the selected point (POI) and all classes appear once in the legend (you should not call legend here as it is done in the main function). Should a y value not appear in *colors* use black as default color. Use red for the selected point. Moreover, the size of the markers should be the weight multiplied by the default marker size.

1.6 Fit Local Surrogate

Finally, fit a decision tree with training data and weights. Return the fitted tree in the function *fit_explainer_model*. What could be problematic?

2 Anchors

In this section, we use the framework anchor-exp to explain instances and to visualize them inside our plots.

Associated file: *tasks/anchors.py*.

2.1 Retrieve Explanations

Use *AnchorTabularExplainer* in *get_explanation* to explain an instance. In our case, an instance is a selected point. Have a look into the API and work with the returned explanation.

Because the framework returns human-friendly text only, we have to prepare the data for our plot. Use regex to extract the information into a dictionary. Following patterns with A as feature name placeholder and v_- and v_+ as lower and upper values are possible:

- $A < v_+$,
- $A > v_-$,
- $v_- < A < v_+$.

Since also $\leq$ and $\geq$ appears in the patterns, simply replace them with $<$ or $>$, respectively. Return the dictionary s.t. each feature name is mapped to a tuple of lower and upper bound. However, make sure all labels are included and all lower and upper bounds are set.

2.2 Plot Anchors

The precision, coverage and bounds from the previous function should now be used to visualize an anchor in *plot_anchor*. Add four lines by using *plt.plot* and a text with precision and coverage. Think about the meaning of precision and coverage.